Hakin9
ON DEMAND

Vol.3 No.1
Issue 01/2014(16) ISSN: 1733-7186

WIRESHARK

WIRESHARK TIPS AND TRICKS

SNIFFING AND RECOVERING NETWORK INFORMATION USING WIRESHARK

TRAFFIC ANALYSIS AND CAPTURE PASSWORDS DETECT/ANALYZE SCANNING TRAFFIC USING WIRESHARK

Dr.Web 9.0

for Windows — the rapid response anti-virus

1. Reliable protection against the threats of tomorrow
2. Reliable protection against data loss
3. Secure communication, data transfer and Internet search

www.drweb.com

Free 30-day trial: https://download.drweb.com

New features in Dr.Web 9.0 for Windows: http://products.drweb.com/9

FREE bonus — Dr.Web Mobile Security:
https://download.drweb.com/android

Wireshark

Table of Contents

Dear Readers,

We are happy to present you completlely new issue dedicated to the most known sniffer – Wireshark. We are sure all of you know this special toll. You can use it to analyze network traffic, intrusion detection, or communication protocols development.

This issue is a guidbook for all those who wants to learn step-by-step how to use this sniffing tool. With this issue you will get basic knowledge on how to start an amazing adventure with Wireshark, but you will also dive into deep waters of hacking knowledge. Except of BASICS section you will also find TRAFIC ANALYSIS and INTRUSION DETECTION sections, full of our expert's tutorials.

We would also thank to our friends from PenTest Magazine for sharing their great articles. We appreciate their work which helped us to create this great issue.

Enjoy!

Regards,

Ewelina Nazarczuk

Hakin9 Magazine Junior Product Manager

and Hakin9 Team

Editor in Chief: Ewelina Nazarczuk
ewelina.nazarczuk@hakin9.org

Editorial Advisory Board: Bamidele Ajayi, Kishore PV, Gilles Lami, Rodrigo Comegno, Tim Singletary, Jarvis Simpson, Dallas Moore, Hammad Arshed, Gregory Chrysanthou, Elia Pinto, Jeff Smith, Arnoud Tijssen, Tom Updegrove and others.

Proofreaders: Jeff Smith, Krzysztof Samborski

Special thanks to our Beta testers and Proofreaders who helped us with this issue. Our magazine would not exist without your assistance and expertise.

Publisher: Paweł Marciniak

CEO: Ewa Dudzic
ewa.dudzic.@hakin9.org

Product Manager: Ewa Duranc
ewa.duranc@hakin9.org

Production Director: Andrzej Kuca
andrzej.kuca@hakin9.org

Art. Director: Ireneusz Pogroszewski
ireneusz.pogroszewski@hakin9.org
DTP: Ireneusz Pogroszewski

Marketing Director: Ewelina Nazarczuk
ewelina.nazarczuk@hakin9.org

Publisher: Hakin9 Media sp. z o.o. SK
02-676 Warszawa, ul. Postępu 17D
NIP 95123253396
www.hakin9.org/en

Wireshark Tips and Tricks

by Tony Lee and Jason Bevis

If you were tasked to put together a forensic toolkit with 25 tools or less, chances are Wireshark would be one of those tools--especially if you planned on dealing with packet captures. Because it is free, open source, and cross-platform, Wireshark makes a great packet capture and analysis tool for just about any forensic toolkit. Never the less, this staple tool has been around for so long (think back to the days of Ethereal) that we sometimes take it for granted. In this article we will explore a few tips and tricks that highlight why we like this tool so much.

Obtaining the software

This seems easy enough, right? Many Linux distributions come with Wireshark installed as a default package and Windows has an easy point and click install package. But did you know there is a PortableApps release of Wireshark? How about a U3 release as well?

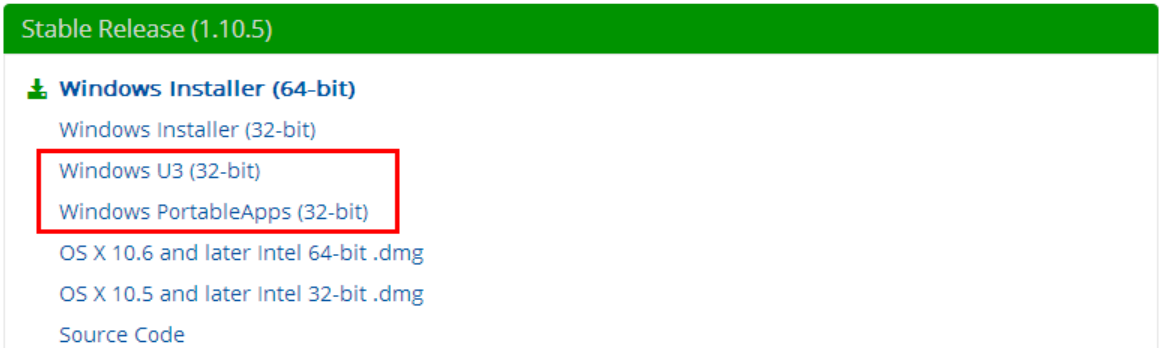


Figure 1. Install options – http://www.wireshark.org/download.html

The PortableApps and U3 downloads allow you to run Wireshark from a USB stick without the needing to install the software on the listening machine. Instead, when you insert the USB stick, you are good to go. There are some caveats that exist so be sure to read the fine print. In any event, this provides a flexible and portable option for running Wireshark on other machines.

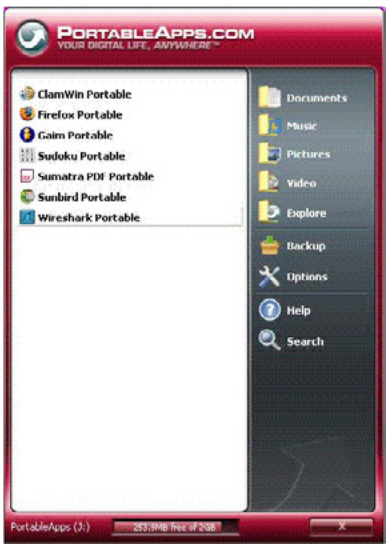


Figure 2. PortableApps Wireshark

Figure 3. U3 Wireshark

Working with pcaps

Again, this is another topic that seems self-explanatory. However, when we teach classes on capturing and analyzing traffic we seem to get this question: “I have several interfaces, how do I know which interface in which to listen?” The reason for this question stems from the main screen, which allows users to select an interface, but it does not show which interface(s) are seeing traffic. The work around is to click capture → Interfaces. This menu option shows you the interfaces in real time so you can see which are live and receiving traffic.

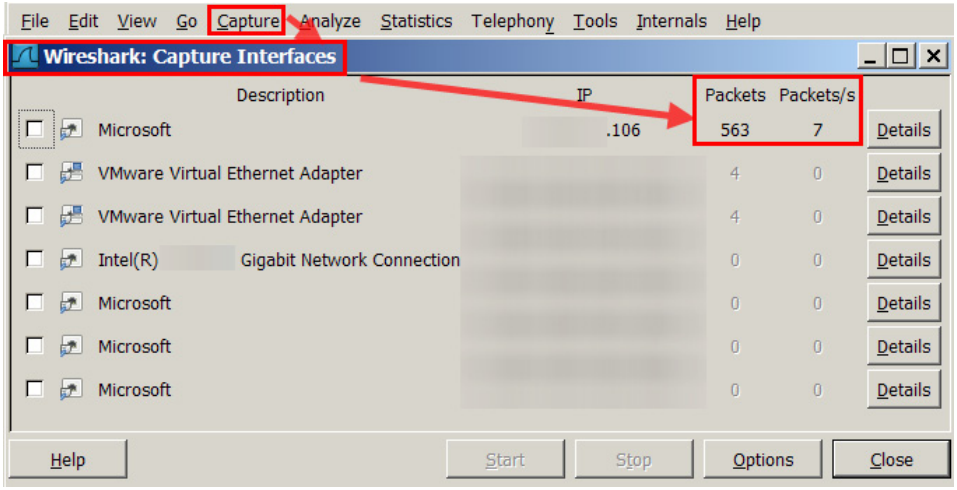


Figure 4. Capture Interface screen showing live traffic

Opening pcaps that were saved or created in other programs is as easy as dragging and dropping them into Wireshark, but did you know you can easily merge pcaps by doing the same thing? By default dragging and dropping multiple pcaps into Wireshark will cause it to merge the pcaps chronologically. You can also merge pcaps by going to File à Merge where you can select different options to merge the pcaps (prepend, append, or chronologically). This is useful if you collect from multiple sensors or interfaces and want to see the complete picture.

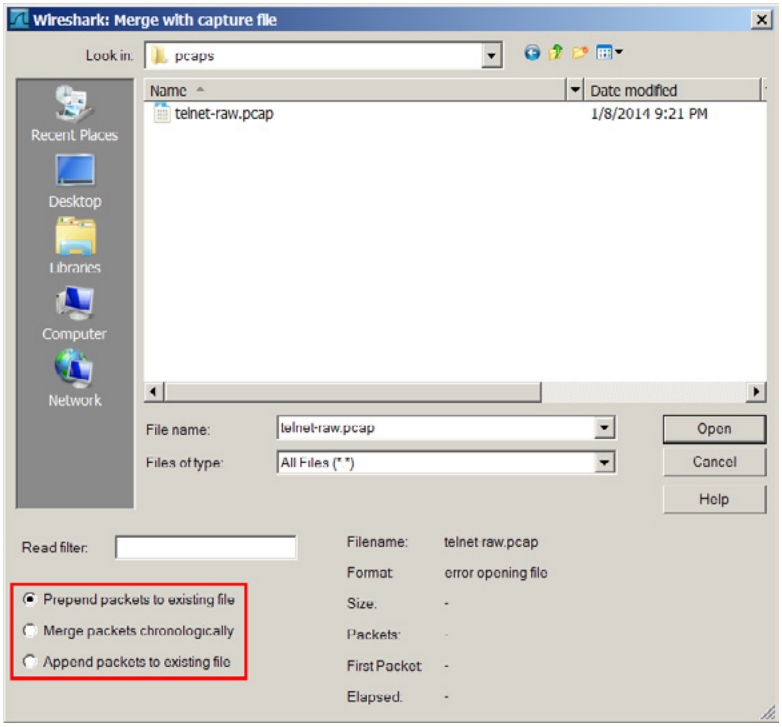


Figure 5. Merging pcap options

Need a few pcaps to work with? How about viewing some more interesting protocols than what is most likely found on your home network? The wiki at Wireshark.org has lots of pcaps to pick and choose from: *http://wiki.wireshark.org/SampleCaptures*.

Display filters

Since traffic on a busy pipe can be overwhelming, Wireshark provides the capability to use capture and display filters. However, in most cases, for troubleshooting or quick analysis it is best to capture unfiltered traffic and then identify packets of interest using display filters. It is inevitable though that sooner or later you will have to learn a few basic display filters. Start off with easy ones such as the following:

Display only certain protocols:	Display only certain addresses:	Combine Display Filters
Examples: http telnet ftp	Examples: ip.addr == 172.16.100.11 ip.src == 172.16.100.11 ip.dst == 172.16.100.11	Examples: ip.addr == 172.16.100.11 && http dns || http

The simple filters above should be enough to meet most basic requirements, however if a more complex display filter is needed, the Wireshark Expression button is very helpful. It is located right next to the display filter field and acts as a sort of a wizard for building display filters.

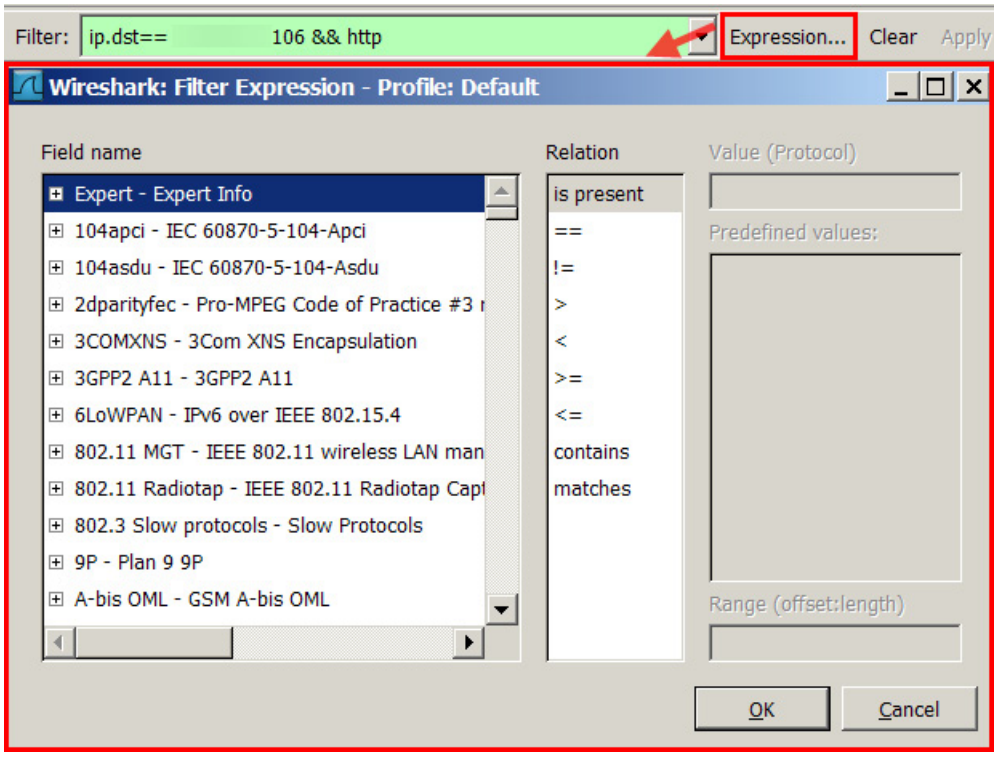


Figure 6. Display filter Expression button

You can also check out this page for more good ideas: *http://wiki.wireshark.org/DisplayFilters*.

High to low level

At times you may be given a pcap with no background knowledge about the protocols or data captured. One of the easiest ways to gain a quick understanding of the situation is by using Wireshark's statistics features. Our favorite option for drilling down on protocols is Statistics → Protocol Hierarchy.

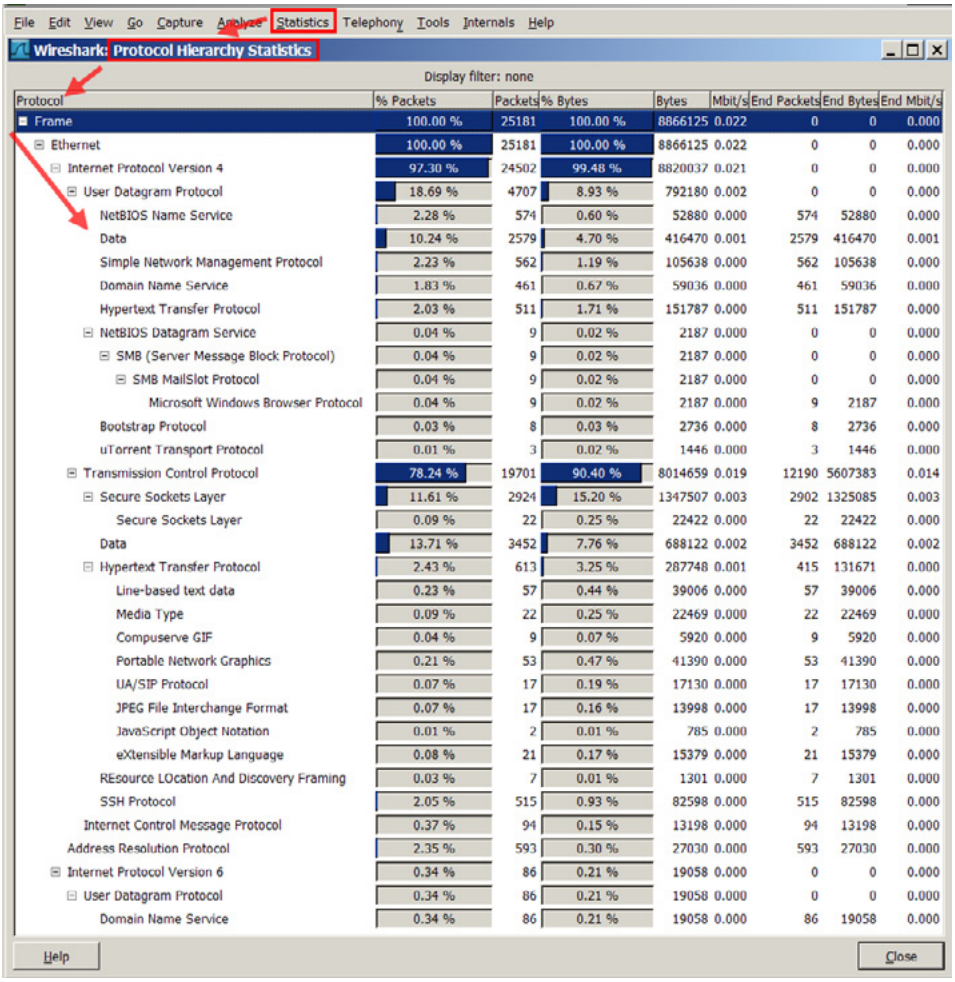


Figure 7. Statistics → Protocol Hierarchy displays an OSI breakdown

Not only does this give you an excellent OSI breakdown, but you can right click on a protocol to apply a filter to look at only those packets of interest.

Extracting files

There are plenty of methods/tools to extract files from a pcap, including: foremost (*http://foremost.sourceforge.net*) and Network Miner (*http://sourceforge.net/projects/networkminer*) – however, Wireshark can also be used to extract files. Some would say that you have to follow the stream and export raw bytes to extract a file. While this is one possible method, depending on the protocol, you may still have to use a hex editor to clean up the resulting file. Wireshark can extract objects from supported protocols by using: File à Export Objects → <Protocol>. In our example, we are exporting from HTTP – which is very common. It makes exporting binaries, zip files, images, and even JavaScript and applets easy.

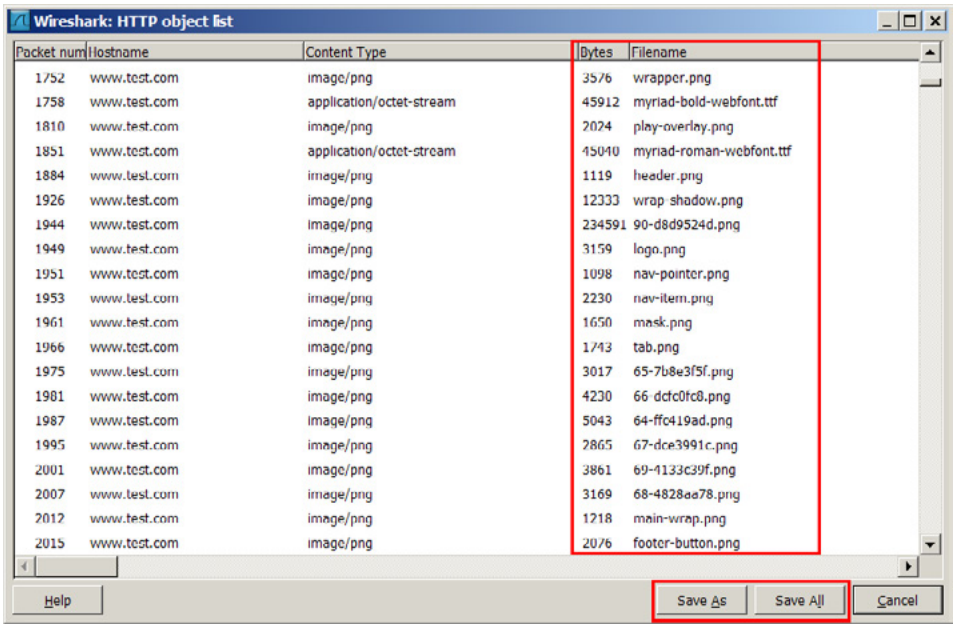


Figure 8. Exporting HTTP objects from Wireshark

Generating firewall rules

Lastly, an interesting feature of Wireshark is that it can generate firewall rules (for different vendors) so you can prevent further unwanted traffic from traversing your network boundary. Just select a packet of interest and click Tools → Firewall ACL Rules. The product drop down allows you to select from the following vendors:

- Cisco IOS (standard)
- Cisco IOS (extended)
- IP Filter (ipfilter)
- IPFirewall (ipfw)
- Netfilter (iptables)
- Packet Filter (pf)
- Windows Firewall (netsh)

Then select the IP of interest and decide which IP address, inbound or outbound, and if you want to deny or permit the traffic.

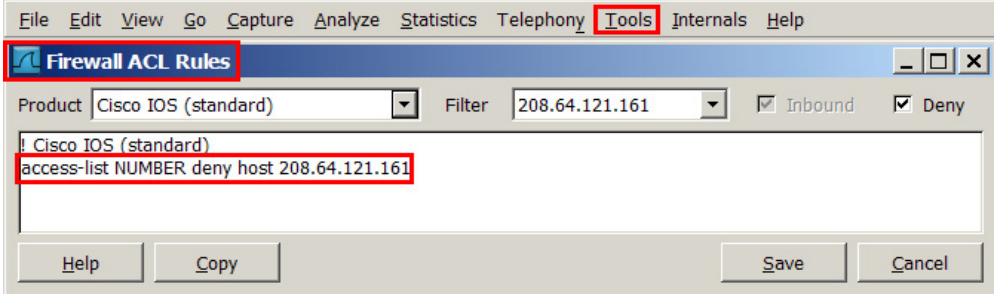


Figure 9. Tools → Firewall ACL Rules

Final thoughts

Wireshark is one of those tools that has been around so long we may take it for granted at times, but when you need something to capture or carve packets it is freely available, reliable, and capable. Hopefully this article has either cleared some cobwebs for you or possibly served as a brief introduction to some of the key (and sometimes under advertised) features of the tool. Feedback is always appreciated using our contact information below. Thanks for reading--now go carve some packets.

About the Author

Tony Lee has more than nine years of professional experience pursuing his passion in all areas of information security. He is currently a principal security consultant at FireEye Labs, in charge of advancing many of the network penetration testing service lines. His interests of late are Citrix and kiosk hacking, post exploitation tactics, and malware research. As an avid educator, Tony has instructed thousands of students at many venues worldwide, including government, universities, corporations, and conferences such as Black Hat. He takes every opportunity to share knowledge as a contributing author to Hacking Exposed 7, frequent blogger, and a lead instructor for a series of classes. He holds a Bachelor of Science degree in computer engineering from Virginia Polytechnic Institute and State University and a Master of Science degree in security informatics from The Johns Hopkins University.

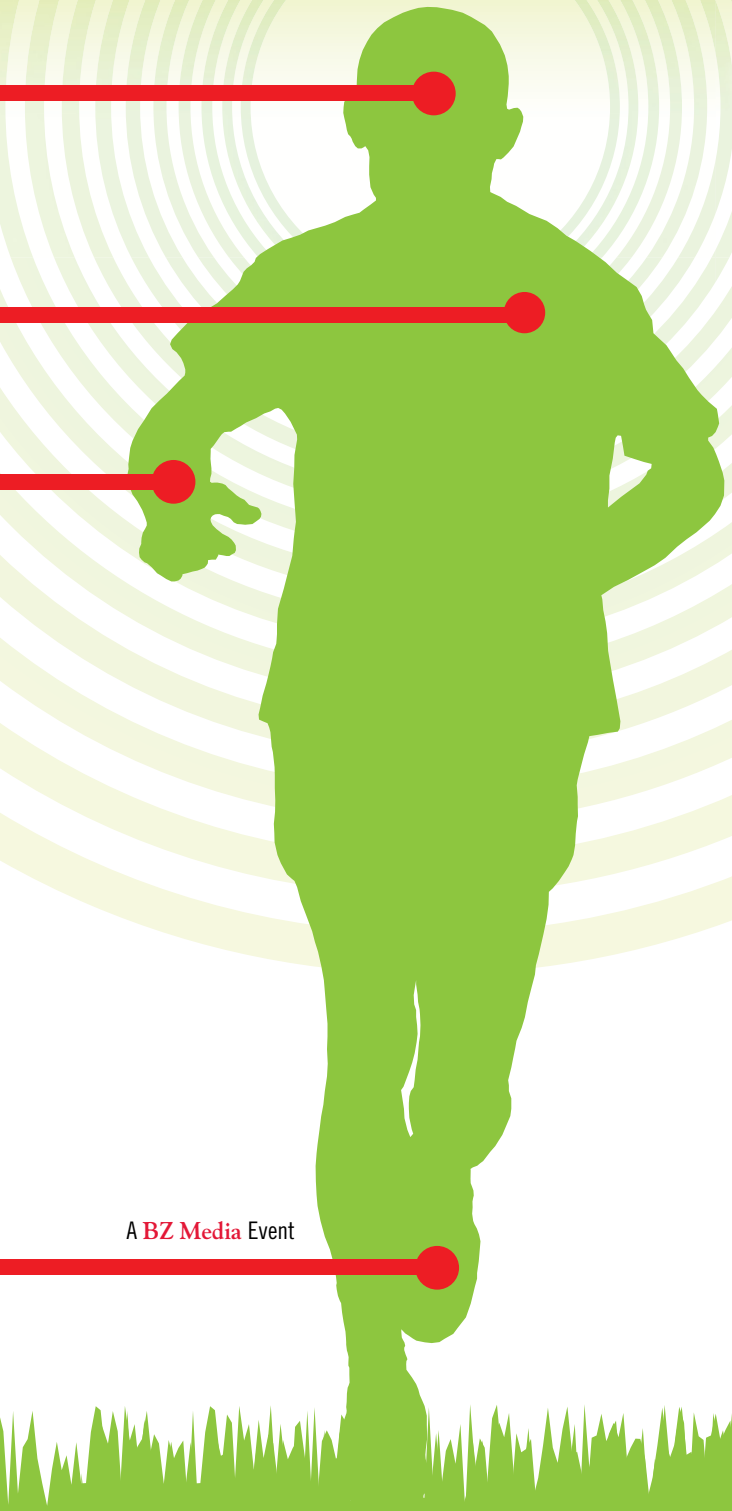
Email: Tony.Lee -at- FireEye.com
Linked-in: http://www.linkedin.com/in/tonyleevt

About the Author

Mr. Bevis has an outstanding track record in consulting services, strategic security solutions, incident response, and client management. Jason is a results driven leader with over 17 years of experience including more than six years of "Big X" Consulting experience. He is an expert in the field of information security, incident management, and risk management with a strong technical background in incident response, forensics, security strategy, and network and system architecture. He runs his personal security blog and his interested of late are within the maker community playing around with Arduino projects and researching sensor communications.

Email: Jason.Bevis -at- FireEye.com
Linked-in: http://www.linkedin.com/in/jasonbevis

Join the Wearables Revolution!



Wearables DevCon

A conference for Designers, Builders and Developers of Wearable Computing Devices

Wearable computing devices are the Next Big Wave in technology. And the winning developers in the next decade are going to be the ones who take advantage of these new technologies EARLY and build the next generation of red-hot apps.

Choose from over 35 classes and tutorials!

- Learn how to develop apps for the coolest gadgets like Google Glass, FitBit, Pebble, the SmartWatch 2, Jawbone, and the Galaxy Gear SmartWatch
- Get practical answers to real problems, learn tangible steps to real-world implementation of the next generation of computing devices

March 5-7, 2014
San Francisco
WearablesDevCon.com

A BZ Media Event

Getting Started with Wireshark

by Sebastian Perez

As a pentester, I always get involved in different projects from different clients and no matter what the objective is, having the knowledge and the proper tool to perform the task will save a lot of time, and avoid some headaches. This article will try to aid for those scenarios where a network analysis should be performed. We will focus in one of the most important tools for a pentester: Wireshark.

For most of the engagements a pen tester could perform, there is always a network component, and being able to see, analyze and store all the network transactions is essential to understand network behaviors and evidence all the performed tasks. For such objectives, Wireshark is what was promised, and more. Looking for a formal definition of Wireshark, as stated in the official website (http://www.wireshark.org/faq.htm), it is a free open-source network protocol analyzer. What does this mean? It means that Wireshark will capture all the traffic it can hear from the selected network interface, parse it and present it to the user in a friendly manner. Within the captures packets, Wireshark will allow us to perform several tasks, such as analyze network problems, get network statistics, and even detect network intrusion attempts. Wireshark runs in most of the operating system available in the market, including Windows, OS X, Linux and UNIX, and it has two different interfaces, allowing users to adapt it to their own requirements; it could be executed in a GUI (Graphic User Interface) or CLI (Command Line Interface). The installation process is really simple, no matter if it's being performed on Windows, or *nix based systems. Besides Wireshark, the Winpcap library (libpcap in *nix) will be needed to be installed also. If the Windows GUI version is executed, the following screen will be presented:

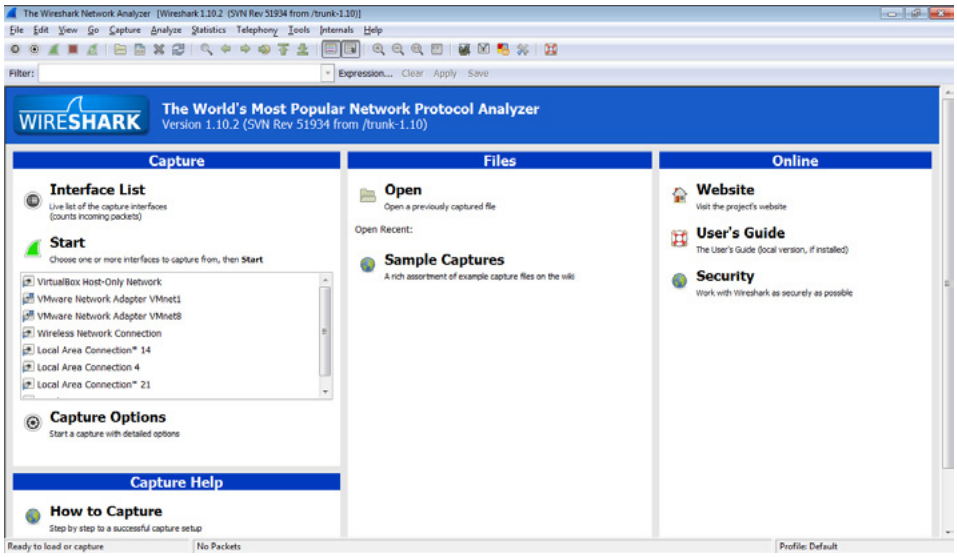


Figure 1. Wireshark main window

The main screen is divided in 3 tabs. The one in the left side is related to the capture options, and the list of interfaces in which Wireshark may be able to listen and capture traffic. The middle tab shows the saved Wireshark sessions; and the right one is for the online content, such as the user guide and official website. We will focus on the Capture tab, as the other two are self-explanatory.

In order to capture traffic, we need to specify to Wireshark which network interface(s) we would like to listen. Currently, most computers have more than one network interface, so in case we are unsure of which is the proper one to listen, Wireshark provides an interface list. This option will show all the network interfaces in the computer and the packet count on each of them (The count starts when we access this option). This will make easier to identify the active interfaces, and probably the one with the most count of packets is the one we would like to capture. Clicking on "Details" button, will provide even more information of each network interfaces.

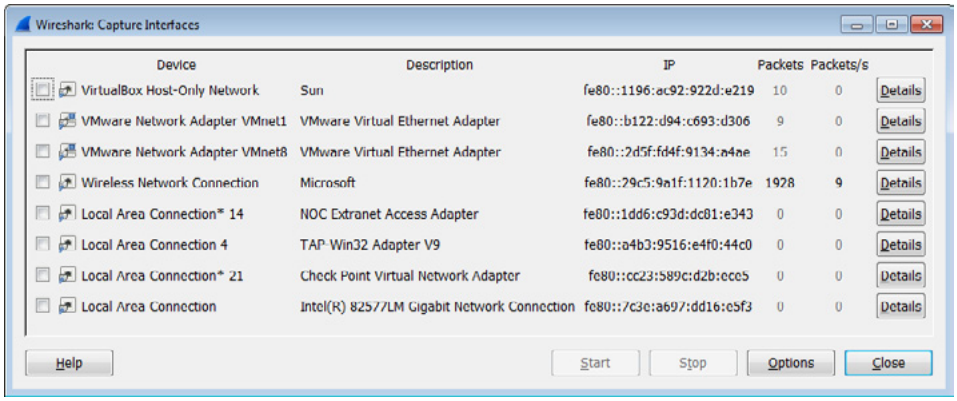


Figure 2. Example of the list of interfaces to capture

Once proper network interface is identified, just select it in the main screen and click on the "*Start"* button. Once the Wireshark starts to capture, will show a screen like the following: Figure 3.

This screen is divided in 3 rows, each of this showing different information. The top row is the packet list pane; the middle row is the packet details pane and the bottom one is the packet bytes pane. Let's talk about each one of them:

Packet list pane (1)

This pane displays all the packets that are being captured during the current session; in case of a previously stored session, it will display all the packets that were saved on that session. Each of the lines within this pane corresponds to one packet that was captured. By selecting one of them, the packet details pane and packet bytes pane will be updated to reflect the content of the selected packet. The default pane configuration contains 7 columns, displaying the following attributes:

- *No.:* The number of the packet in the capture file. This number is related to the current session only, and is incremented by one for each packet.
- *Time*: The timestamp of the packet. The default configuration shows the amount of seconds since the beginning of the current session. This value could be changed to reflect the proper date and time instead, such as the UTC time.
- *Source*: The IP address from where this packet came from. In case that no IP address was available (ARP packets, for example), the name and part of the device's MAC address is displayed.
- *Destination*: The IP address where this packet is going to. In case of Broadcast packets, the legend "*Broadcast"* is displayed. And similar to "Source" field, if no IP address was available, the name and part of the device's MAC address is displayed.
- *Protocol*: Display the protocol used, in a short name version, or abbreviation, if a short name is not available
- *Length*: Display the packet length
- *Info*: Provides additional information about the packet content, such as TCP header fields

More columns could be added as required, depending on the information the user may need to gather, as Wireshark provides the functionality to add custom columns, using the packet fields available. In order to add a column, just right-click on a column name and select the option "*Column preferences"*. A dialog window will show up with a drop-down list, containing a list of predefined columns. Within this list, a "*custom"* option is available to define personal filters.

Going back to the packet list pane, by right-clicking on this window, it is possible to access a menu of tasks to perform, including options to filter packets, follow streams, and copy the packets in TXT or CSV formats. One of the functions that can be very helpful if the user is working with TCP or UDP protocols is the *Follow TCP Stream* or *Follow UDP Steam*. This function will display the data from a TCP or UDP stream in the way that the application layer understands it. By selecting this function, an automated filter will be applied within the packet list, and a dialog window will show a reconstruction of the messages that were sent and received within this stream.

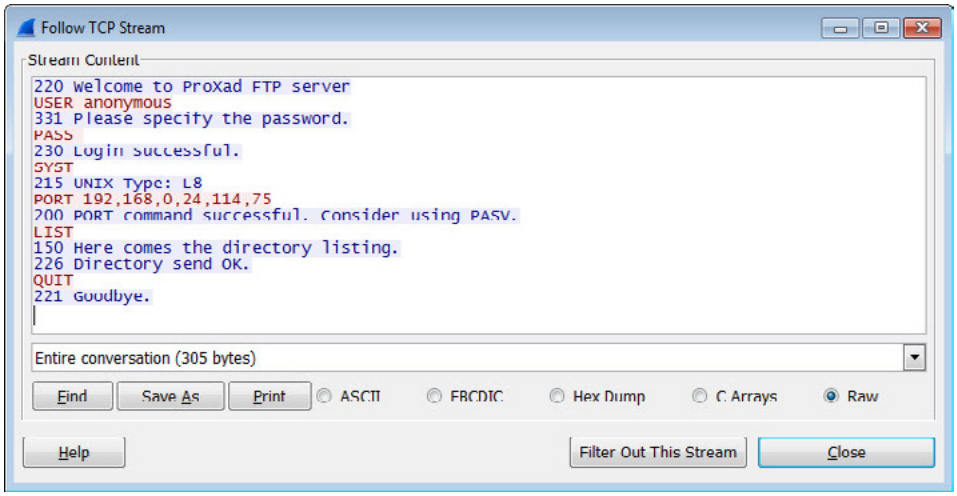


Figure 4. Following a Telnet stream

This window will display the requests to the server in red color, and the responses from the server in blue color. This function is very helpful if the user wants to find some clear text within a particular connection, such as user credentials, or even website cookies. Within this window, there is a search button, to find a particular string within the stream. Also, it is possible to change the character representation between ASCII, EBDCID, hexadecimal, C arrays, or even the raw data

Packet details pane (2)

This pane shows the protocols and protocol fields of the selected packet using a tree structure, which can be expanded and collapsed. Every tree parent is related to a different network layer. This pane will disassemble the packet and display the content of the different layers that compose it. It will parse and show the values of the different fields in each of the protocols involved. In this structure, information like the MAC address of the devices involved and the source and destination port could be observed. By right-clicking on this pane, it is possible to access a menu of tasks similar to the one displayed in the packet list pane.

Packet bytes pane (3)

This pane is divided in 3 different columns to show the raw data of the selected packet, using a hexadecimal notation:

- The left column displays the offset in the packet data;
- The middle column displays the packet data in hexadecimal representation
- The right column displays the corresponding ASCII codification, or the dot “.” character, if there is no ASCII codification to display.

By right-clicking on this pane, it is possible to change the codification from hexadecimal to binary.

Before starting capturing packets, there are two things to analyze.

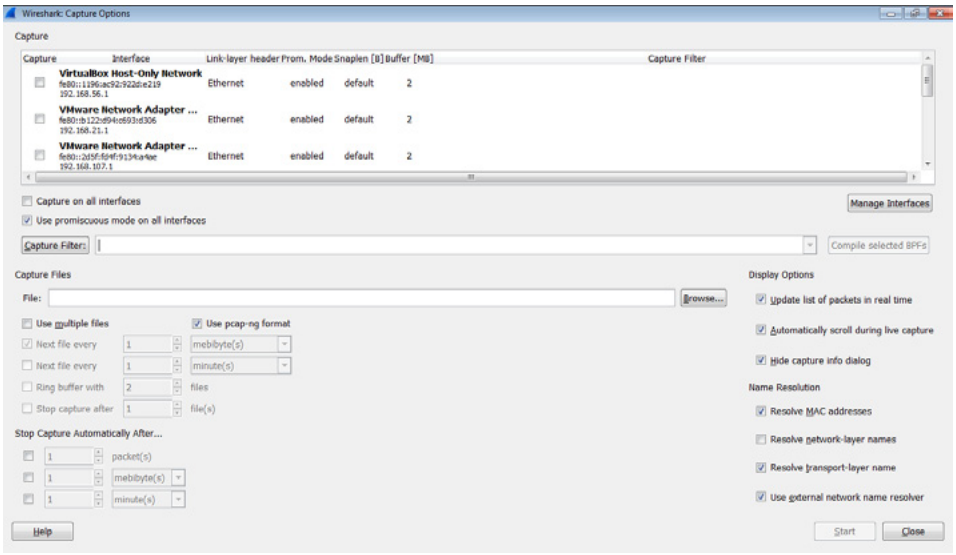


Figure 5. Capture options

The first one I would like to talk about is the capture options, as this allows to specify the behavior that Wireshark will have related to the captures. Let's enumerate the most important options available:

- *Capture on all interfaces*: allows Wireshark to capture on all network interfaces, or to select multiple network interfaces to listen, using the interfaces list that appears above
- *Use promiscuous mode on all interfaces*: this option allows specifying if Wireshark should activate the promiscuous mode setting in all the network interfaces. We will review this setting later.
- *Capture Filter:* this field could be used to specify a filter to be applied in all the selected interfaces to listen. This filter will prevent unwanted packets to be captured and stored in the current session; if this field is not completed, no filters will be applying to the interfaces. By clicking in the "Capture filter" button, a dialog will open and show a list of saved filters, giving to the user the alternative to add or delete more filters. This filter is not the same filter that appears in the Wireshark main screen, while a session is in progress. That filter will be explained later.
- *File*: this field could be used to specify the filename where the session will be stored; if this field is not completed, the session will be stored in a temporary file. By clicking on "Browse" button, a dialog will appear, allowing the user to browse the file system and select the destination of the file.
- *Use multiple files*: Wireshark has the ability to store the session across multiple files, depending on the criteria provided. By selecting this, four new options will appear as shown below:
    - *Next file every n mebibyte(s)*: Specify the maximum size in MiB of the capture file. Once this size is reached, a new file is created.
    - *Next file every n minute(s):* Specify the maximum time a capture file will be used. After this time is reached, a new file is created.
    - *Ring buffer with n files:* Specify the number of files that will be part of a ring buffer. This ring buffer allows to rotate the captures within this number of files.
    - *Stop capture after n file(s):* Specify the maximum number of files that Wireshark will use to store the current session. If this value is reached, the capture will stop.
- *Stop Capture Automatically After:* this section contains 3 different options to automatically stop the packets captures.

  - *n packets:* Specify the maximum amount of packets that Wireshark will capture before stopping.
  - *n mebibyte (s):* Specify the maximum amount of MiB that Wireshark will capture before stopping.
  - *n minute (s):* Specify the maximum amount of time that Wireshark will capture packets before stopping.

- *Update list of packets in real time:* this option allows the user to specify if Wireshark will update the packet list pane while the capture is still active, or to not display the packets until the capture is stopped. If this option is enabled, Wireshark will use one process to capture the packets and a different process to display the packets in the packet list pane.
- *Automatically scroll in live capture:* this option allows the user to specify if Wireshark will perform an automated scroll while new packets are being captured. If this is not enabled, the new packets will not be shown in the packet list pane until the user scroll down.
- *Hide capture info dialog:* this option allows the user to specify if the info dialog will be displayed meanwhile a capture is in progress. The info dialog will show statistics of the protocols captured, and the time since the capture started
- *Resolve MAC Addresses:* this option allows the user to specify if the devices MAC address captured should be resolved into names.
- *Resolve network-layer names:* this option allows the user to specify if the network-layer names captured should be resolved into names.
- *Resolve transport-layer name:* this option allows the user to specify if the transport-layer names captured should be translated into protocols.
- *Use external network name resolver:* this option allows the user to specify if the name resolution will be performed through DNS lookups or not.

The second thing I would like to analyze before starting to capture packets is the necessity of configuring the network interface in promiscuous mode. Let's start with a short review of what is promiscuous mode and then identify whether this is needed or not. If we consider a network environment connected through a hub device, all the packets that came from one host will be sent to all the other hosts within the same network (similar to what happens in a wireless network environment). Every host that receives the packet compares the destination MAC address of the packet to the MAC address of its own network interface. If both MAC addresses match, then the network interface captures the packet and processes it. If the MAC addresses do not match, then the packet is discarded. If the promiscuous mode is enabled, every packet that arrives to the network interface, no matter what its MAC address is, it will be captured from the network. This allows us to capture all traffic that travel in the network. Now that we defined the promiscuous mode, let's use a more realistic scenario. Today it is not common to find hubs connecting networks; even it is getting difficult if you want to buy a hub in a computer store. Today's network uses switches to transmit messages only to the destination, avoiding flooding the network with unnecessary packets, and making it more difficult to sniff traffic. With a switching network, enabling the promiscuous mode will not have any effect, as we are not receiving traffic that is not intended to us. Considering this scenario, if we want to capture all the packets that travel across the network, we probably need to do one of the following: perform and ARP poisoning or connects to a mirroring port. These topics are out of the scope of this article, but I wanted to give a real world scenario before continuing.

Going back to our topic, the main concern is if it is required to listen in promiscuous mode, or not.
There is not correct answer, as it depends of the nature of the tasks to be performed. In case of a network administration that wants to inspect all the traffic that travel across the network, then yes, the promiscuous mode will be necessary. But, if we consider a penetration tester that only wants to identify the traffic between his computer and a website, or a particular server, then this mode is not required, and enabling it will provide a lot of unnecessary traffic.

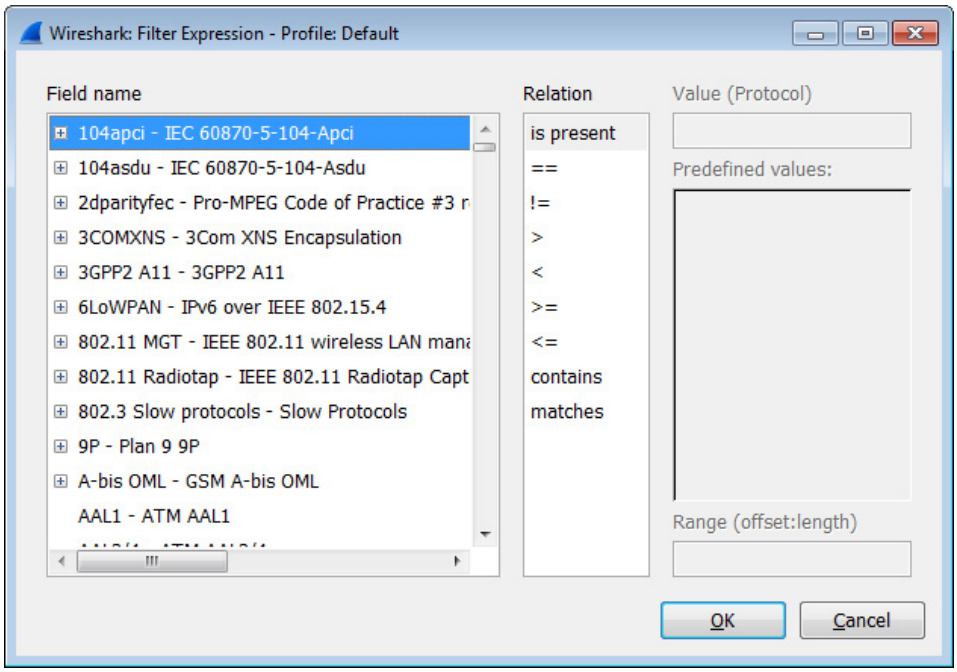


Figure 6. Filter expression

Now that we have enough information to adapt the Wireshark to our necessities, it is time to start the sniffing. Once the capturing process is running, the packet list pane will display a lot of traffic, probably more than the desired one. This is when another of the big features of Wireshark comes to play. This tool provides an extensive set of filter options, allowing the user to display only the packets that he wants to see. The filter toolbar is above the packet list pane. By clicking on the "*Expression"* button, a dialog box will open with a complete list of filters that Wireshark is able to manage.

This list is a very good point to start for those who do not have expertise using Wireshark, as the user can easily select the fields he wants to use in the filters; and it is an excellent way to learn how to write those filters. This dialog is composed of 5 fields, as follows:

- *Field Name*: this field contains a list of protocols in a tree structure. Every protocol that has fields that could apply as filters are displayed at the top level of the tree. By expanding the protocol tree, the user will be able to get a list of the field names available for that protocol.
- *Relation*: this is the operator that will apply to this filter. The operators could be logical or comparison. Once a relation is chosen, the user will be able to complete one of the following 3 fields, related to the values.
- *Value*: this field contains the value that should be compared within the one in the packet, in order to Wireshark be able to match it. Between will appear the type of value (Character, string, number, etc.).
- *Predefined values:* for particular protocols and fields, there is a list of predefined values from where the user must select which is the desired one.
- *Range*: for some protocols it is possible to add a range of values to match with.

Within the filter toolbar there are 3 buttons that perform the following:

- The "*Clear"* button removes all the filters applied to the packet list pane.
- The "*Apply"* button use the filter typed in the text box.
- The "*Save*" button store the filter for a future use.

The text box in the left side of the toolbar could be used to write and apply filters. Instead of looking through the list of filters, the user could easily write the desired ones, if the proper syntax and names are known. Be aware that these filters are case sensitive; for instance, it is not the same thing writing "*ip*", than "*IP*". The filter contains 3 different items that compose the syntax. There is one field (usually a packet field), one operator and one value. One exception for this syntax is that the operator "is present", does not require a value to be compare with, as it returns true if the field is present within the packet. For example, if we want to filter only packets from or to the IP address 192.168.0.1, I will use the following filter

```
ip == 192.168.0.1
```

There is a list of the most common operators used in the filters

Comparison operators

Operator	C-like	Description
eq	==	equal to
ne	!=	not equal to
ge	>=	greater than or equal to
gt	>	greater than
le	<=	less than or equal to
lt	<	less than
matches		matches to
contain		contains
is present		is present

Logical operators

Operator	C-like	Description
and	&&	AND logical operation
or	||	OR logical operation
xor	^^	XOR logical operation
not	!	NOT logical operation
[...]	[...]	Substring operator

For some popular filters, instead of writing the ports used for that protocol, the user could write the protocol name instead. For example, it can be written *http* in the filter textbox, instead of the following rule (this is assuming that in the current session, there are no other http servers in ports besides 80 and 443)

```
tcp.port == 80 || tcp.port == 443
```

The list of popular protocols that could be used instead of filtering within the ports used is presented:

```
arp, bootp, smtp, pop3, dns, smb, ldap, ftp, icmp, imap, nbss
```

Wireshark also allows the definition of advanced filters that could be used to match specific bytes positions within the required fields. These bytes could be defined by the use of an offset and bytes length. The proper syntax for the advanced filters is shown below, followed by an operator and value to match with:

```
Packet field or protocol[offset:length]
```

For example, if I want to filter all the ip packets that contains in the bytes 10 and 11 the values 0x30 and 0x45, the filter will be

```
ip[10:2] == 30:45
```

The next list contains some examples of useful filters.

- *ip*: displays only ip packets.
- *tcp.port eq 23 or ftp*: displays only packets of ftp and telnet protocols.
- *ip.src==192.168.0.0/24 and ip.dst==192.168.0.0/24:* displays packets sent between hosts in the same network 192.168.0.0
- *ip.addr == 192.168.0.0/24:* displays packets from or to the IP range selected. Be aware that this is not the same than the previous one. This filter will display traffic from the LAN to an external host, and vice versa.
- *smb || nbns || dcerpc || nbss || dns:* displays packets that is usually associated with Windows hosts.
- *ls_ads.opnum==0x09:* displays packets associated with the Sasser worm.
- *eth.addr[0:3]==00:06:5B:* displays those packets where the MAC address starts with 0x00, 0x06 and 0x5B. This is useful to filter packets that came from a particular vendor's network interface.
- *http.request.uri matches "le.com$":* displays HTTP packets where the last characters in the URI field are "le.com".
- *not broadcast and not multicast:* displays all packets, except broadcast and multicast.
- *ether host AA:BB:CC:DD:EE:FF:* displays all Ethernet packets that goes to and from the network interface that has that MAC address.

Now that we know how to capture traffic and filter all the undesired packets, it is time to see what we could do with the traffic captured. Wireshark includes a sophisticated traffic analyzer and decoder. As more devices are being connected to personal and business networks, traffic analysis is becoming more important. From identifying a network misuse, to detect an intrusion, the ability to analyze the traffic is crucial for network administrators and security officers. Wireshark provides several functionalities for a better understanding of the traffic captured. These functionalities could be found within the toolbar, in the "*statistics"* menu. The most important ones will be explained below:

- *Summary*: this function provides general statistics about the current session. This information includes the timestamps of the first and last packets captured, information about the file where the session it is being saved, and statistics about all the traffic captured, such as amount of packets, average packets per second, and average bytes per second.
- *Protocol Hierarchy*: this function displays a tree structure of all the protocols captured in the current session. The hierarchy is based on the OSI network model. It will also provide some statistics of amount of packets and size per protocol. Be aware that usually, each packet has more than one protocol (each of them at a different OSI layer) and there are some situations where the same protocol could appear more than once in the same packet. Wireshark will count each of these occurrences as different packets.
- *Conversations*: this function displays a list of all the conversations captured and statistics for each of one, within the current session. A conversation is the traffic that is sent and received between two specific endpoints. The available tabs within this dialog box will be enabled depending the type of traffic captured and will show the number of conversations captured in the label; if no conversations were captured for a specific protocol, the tab will be greyed out.
- *Endpoints*: this function displays a list of all the endpoints captured and statistics for each of one, within the current session. An endpoint is each end of a conversation, which means that for each conversation we have identified, there are two endpoints. The available tabs within this dialog box will be enabled depending on the type of traffic captured, and will show the number of endpoints captured in the label; if no endpoints were captured for a specific protocol, the tab will be greyed out.
- *IO Graphs:* This function displays a customizable graphic. It has up to five different filters that could be applied to be represented in the graphic with a different color. By clicking on the graph, will show the selected packet in the Wireshark main window.

- *WLAN Traffic:* This function displays statistics of the captured wireless traffic within the current session. In the network overview section, statistics for each of the wireless networks (BSSIDs) will be displayed. By selecting a particular network, this will display information for each client connected to that wireless network and statistics for each of these.

Another good feature of Wireshark is the ability to intercept and reassemble VoIP packets. This functionality is located in the "*Telephony*" menu within the toolbar. This menu contains different VoIP protocols that could be used to filter packets in the packet list pane, and display only the desired one. Wireshark currently supports the following VoIP signaling protocols, including:

- *SIP*: Session Initiation Protocol (SIP) is a signaling protocol used to set up, manage and end VoIP calls, and multimedia conferences. This protocol includes methods such as INVITE and ACCEPT, and responses that indicate if the call was accepted, if it needs to be redirected, or error codes to indicate an abnormal situation.

- *H323*: is a standard that defines the protocol to provide call signaling and control for multimedia communication sessions.

- *ISUP*: ISDN User Part is part of the signaling protocol SS7, used to establish and release phone calls within the PSTN (Public Switched Telephone Network).

- *MGCP*: Media Gateway Control Protocol (MGCP) is a protocol used to control media gateways that lay on IP networks and are connected to the PSTN.

- *Unistim*: Unified Networks IP Stimulus is a telecommunications protocol designed by Nortel, used in communications between IP Phones and IP PBX (Private branch exchange).

Wireshark also supports the RTP protocol (Real-Time Transport Protocol) which is the protocol used to actual transmits packets in real time over an IP network. The type of packets supported by this protocol includes audio, video or data, over multicast or unicast network services. It is commonly used in streaming, telephony and even television services. It is used within RTCP (RTP Control Protocol) which monitor transmission statics and quality of service. Now, going back to the "*Telephony"* menu, the user will see a list of protocols available to select. Once a protocol is selected, a new window will appear, displaying, in most cases, the streams or packet counters depending on the selected protocol. This menu also contains one option that is "VoIP Calls". By selecting this option, a new window will appear, listing all the calls that were found during the captured session. Different attributes of the calls will be displayed, such as:

- *Start Time:* start time of the call.

- *Stop Time:* stop time of the call.

- *Initial Speaker:* the IP address from the device that initiated the call.

- *From*: depending on the signaling protocol used, it will display a telephone number or a terminal ID.

- *To*: depending on the signaling protocol used, it will display a telephone number or a terminal ID.

- *Protocol*: the protocol used to signal the call.

- *Packets*: number of packets that were sent and received during the call.

- *State*: the current state of the call. This value could be:

  - CALL SETUP: *indicates that the call is being setup.*

  - *RINGING: only for MGCP protocol*, indicates that the call is ringing.

  - IN CALL: indicates that the call is in place.

- CANCELLED: indicates that the call was canceled before being connected.
- COMPLETED: indicates that the call was ended normally.
- REJECTED: indicates that the call was rejected by the addressee.
- UNKNOWN: indicates that the call is in unknown state.

- *Comment*: additional comments for the call. If the protocol H323 is being used, this field will indicate if Fast Start or/and H245 Tunneling is being used.

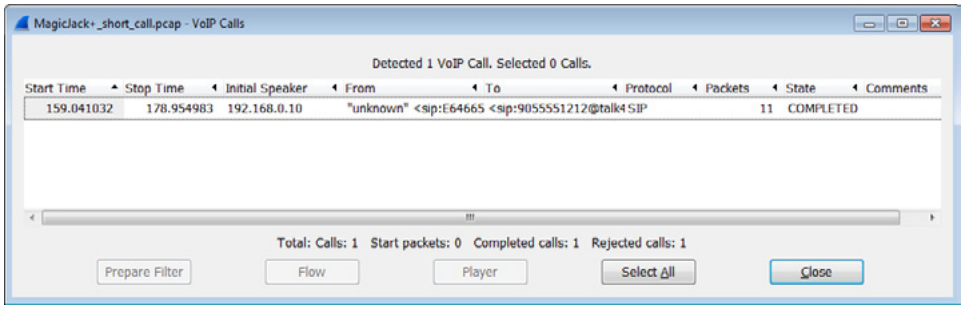


Figure 7. VoIP Call using SIP protocol

It is possible to filter all the packets for a particular call in the Wireshark packet list pane. In order to do so, within the VoIP window, the user just needs to select the desired call and click on the "*Prepare Filter"* button. Wireshark will automatically filter only the traffic that is being involved in that particular call. The "*flow"* button will present a graph analysis of the selected call. This analysis includes which packets were sent and received, at what time this was done, and which protocols were used in each of these packets.

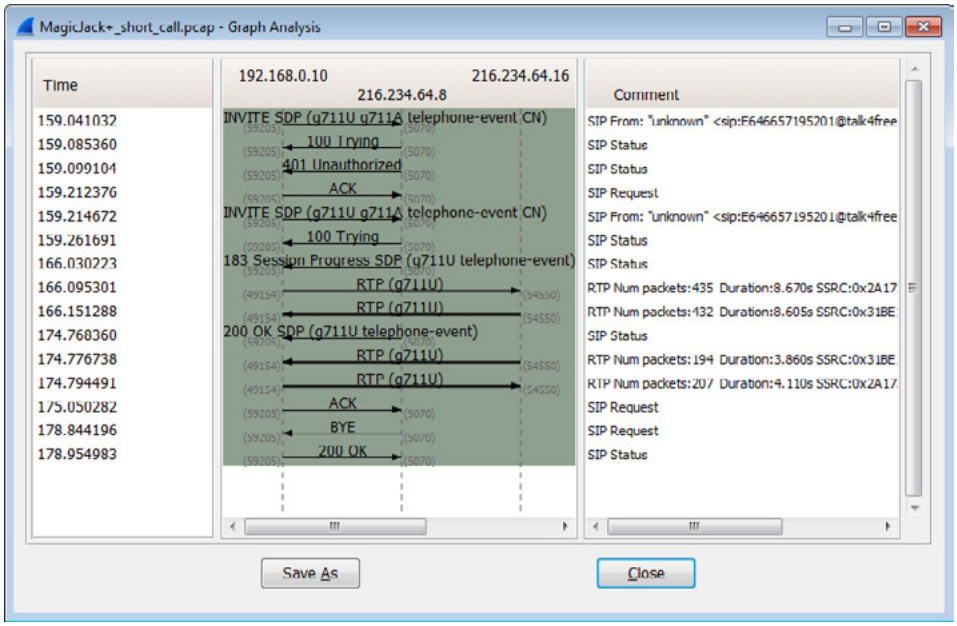


Figure 8. VoIP Graph Analysis

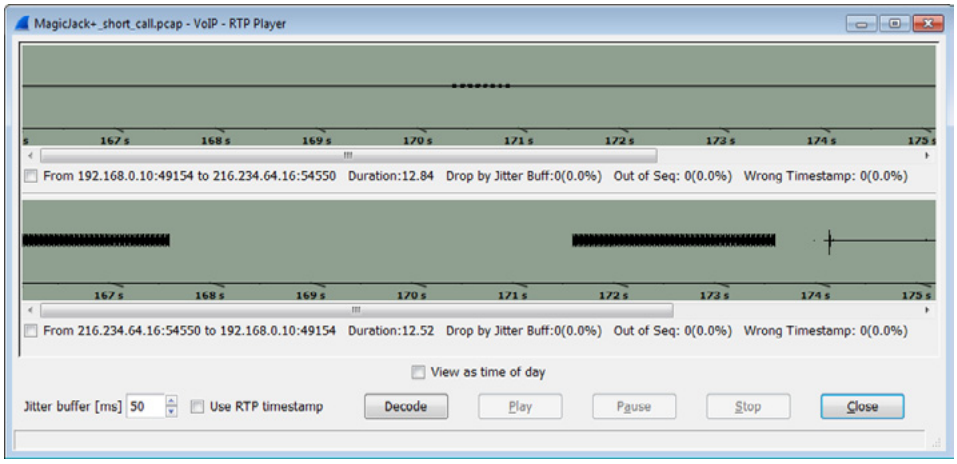


Figure 9. Playing VoIP captured packets

Wireshark also provides a functionality to play the RTP packets captured, that could be accessed from the "*Player"* button. This option will open a dialog window displaying the actors involved in the call in separate lines and the user could play the reconstruction of the call from each part in a separate way or all the parts together (Figure 9).

If you have captured RTP stream packets, but not the SIP packets, it is probably that Wireshark may not recognize the traffic as RTP. For this scenario, Wireshark provides functionality, allowing the user to try and decode the RTP outside of conversations.

- Finally, but not least important, it's the ability to decrypt SSL (Secure Socket Layer) traffic. As you may know, SSL is a protocol that provides confidentiality and message integrity through a symmetric and asymmetric algorithm, and it is widely used on public and private networks. It can be used to encapsulate application layer protocols such as HTTP, SMTP, FTP, and so on. Let's give a quick overview to the SSL handshake, to understand how Wireshark is able to perform this decryption. The client start a connection to an SSL service by sending a *ClientHello message*, and includes its own SSL client configuration, including the protocol version and cipher settings

- The server receives this message and replies with a *ServerHello* message, including its own SSL server configuration, including the protocol version, cipher settings, and the server certificate

- The client authenticates the server certificate against the certificate authority. It also generates a master key, encrypts it with the server certificate, and sends to it

- The client and the server use the master key to generate a symmetric session key that will be used to encrypt and decrypt the information exchanged, and to verity its integrity. The way the symmetric session key is generated depends on the method used for key exchange (Diffie-Hellman, RSA, ECDH, etc.)

This is a quick explanation of how the SSL handshake is being performed. There are more factors involved, but for the purpose of decrypting traffic with Wireshark, this gives a good overview.

There is one warning I must say before continuing. Wireshark is not able to decrypt ALL SSL traffic that travel across the network. The SSL protocol provides integrity, and there is no way to decrypt its content. There are a few techniques that could be used to crack this protocol, but only works on specific conditions. Wireshark performs the decryption by having the private key that is being used to encrypt the traffic. There are also a couple more of requirements needed to perform this. The communication must use RSA key to encrypt the data, and the capture must include the handshake process (ClientHello and ServerHello requests).

Within this information Wireshark is able to decrypt the SSL traffic. In order to perform this, the user needs to access the Wireshark preferences, under the "*Edit*" menu. A dialog window will appear with a tree structure on the left side. Expand the "*Protocol*" tree, scroll down and select SSL. In the right side of the window, an edit button will appear. By clicking this, a new dialog window will appear; click in "new" within this window; and a new one will be presented, requiring the following information:

- IP address: is the IP address of the server that provides the SSL certificate to the client. Wildcard IP address could be used (0.0.0.0)
- Port: is the port used by the server to provide the encrypted service. Wildcard port could be used (0)
- Protocol: is the protocol that was encrypted by SSL. Considering a web traffic (HTTPs), the protocol encrypted was HTTP
- Key File: requires the location of the key file. The key must be in PEM or PKCS12 formats
- Password: is the password that was used to protect the key file, if any

After this configuration was saved, just start a new capture, and if everything was completed properly, Wireshark will decrypt all the SSL traffic that was encrypted using the selected certificate

As observed, Wireshark is a tool that provides several functionalities to analyze network environments. In this article we just covered the most important ones, but there are a lot more of things to do with this tool. I hope you enjoyed reading this article, as much as I enjoyed writing it. See you in the next article.

About the Author

ar.linkedin.com/in/sebasperez/
Sebastian is a Senior Security Consultant in one of the Big Four, located in Buenos Aires, with an experience of more than seven years providing IT Security services. He is responsible for delivering security solutions; including penetration testing, vulnerability assessment, security audits, security remediation, mobile and web application penetration testing, infrastructure security assessments, network analysis, as well as systems architecture design, review and implementation. He also offers internal trainings related to ATM security and penetration testing Android apps. Prior to joining into his current company, he worked as a systems administrator, security policy manager and IT security consultant. His background gives him a thorough understanding of security controls and their specific weaknesses. Furthermore, he is proficient in multiple security application tools, network technologies and operating systems. Sebastian has a post-degree in Information Security, and is currently working on his master degree thesis related to computer and mobile forensics. He also published a CVE # CVE-2012-4991 related to the Axway Secure Transport software, which was vulnerable to Path Traversal vulnerability.

Become a Big Data Master!

Over 45 HOW-TO, practical classes and tutorials to choose from!

Attend The 3rd Big Data TechCon!

The HOW-TO technical conference for professionals implementing Big Data



Come to Big Data TechCon to learn the best ways to:

- Process and analyze the real-time data pouring into your organization.
- Learn HOW TO integrate data collection technologies with data analytics and predictive analysis tools to produce the kind of workable information and reports your organization needs.
- Understand HOW TO leverage Big Data to help your organization today.
- Master Big Data tools and technologies like Hadoop, MapReduce, HBase, Cassandra, NoSQL databases, and more!
- Looking for Hadoop training? We have several Hadoop tutorials and dozens of Hadoop classes to get you started — or advanced classes to take you to the next level!

Big Data TECHCON Boston

March 31-April 2, 2014

www.BigDataTechCon.com

A BZ Media Event Big Data TechCon

Content-Based Intrusion Detection System

by Mark Sitkowski

Nobody ever broke into a bank's IT system by cracking a user's password. It's not cost-effective to waste computer time on such a pursuit, for the sake of the few thousand dollars that may, or may not be in the user's account.

It's far more cost-effective to persuade the bank to let you have access to its database, via a back door. Then, you have access to all of the bank's resources, for the expenditure of a minimum of effort, and without even having to understand how the authentication system works.

On the other side of fence, when your company's product actually is that bank's authentication system, and which it describes as 'Uncrackable', you have to expect this to be like a red rag to a bull, as far as the world's hackers are concerned.

Every day, dozens of them try to break the algorithm, but none ever succeed, so there is some excuse for the complacency which ensues. However, you soon notice that, for every front door attack, there are over a hundred attempts to totally bypass the authentication system, and get in via a back door.

Now, after you've told the world that the authentication system is uncrackable, it would be rather embarrassing to find that the hackers had decided not to bother cracking it, but had broken into your authentication server, instead, and hijacked your database.

You have no control over how the average bank, securities trading company or whoever uses your product, configures their online access server or ATM machine, but you can lead by example, and make sure that your authentication server, at least, can be made hack-proof.

Easy, right? All you need to do is to buy a device which will alert you, as soon as it detects a hack attempt, and prevent it succeeding.

If, after a few weeks of searching on the internet, and talking to prospective suppliers, you find that nothing on the market will do what you want, what do you do?

You write your own, of course...

Defining the problem

When we set up the infrastructure for our authentication server's website, we did all the right things.

The only open port was port 80, there was no GET permission for cgi-bin, no POST permission for htdocs, all other methods like MOVE, DELETE, COPY etc were disabled, and there were no interpreted scripts, like those written in java, perl, shell or ruby.

The only HTML page was index.html, and the other sixty four pages were dynamically created by the CGI – which was an executable, written in a compiled language. That way, if a hacker ran Wget on our site, he'd have no additional clues as to which page called which CGI, or what any of the HTML variables meant.

Bulletproof.

As far as it went, it certainly was. We had many connections each day, from the usual hopeful hackers, who would try to get in by breaking the authentication algorithm, and from the old-timers and incompetents, who would try buffer overflow, not having heard that that particular method didn't work on modern network applications.

Then, after a few months, things changed, as dozens of more determined hackers, with no life of their own, decided that they could combine distributed denial of service attacks with hack attempts. We were inundated with hundreds of queries, each designed to plant or exploit back doors, inject SQL or exploit vulnerabilities in every file whose name ended in '.php'.

We don't use WordPress, cPanel, Joomla, ccmail or any of the other traditionally exploited software packages, so we were immune to all of these attacks, but it was extremely annoying to watch the server logs scrolling like a Las Vegas slot machine, as every unimaginative hack script repeated the same dumb vector anything from two to four hundred times.

Also, it was eating up our network bandwidth, and making the site respond less quickly than we would have liked, and giving perverted pleasure to some hacker, who was watching hundreds of lines of hack script execute.

The last straw came, one day, when we were hit with a DDOS from an address in the Netherlands. It started about 4am, and continued till 11am, during which time the hacker had thrown over twenty thousand vectors at us, at which point, I manually added a firewall rule to block his IP address.

The hacker continued to bang his head against the firewall till around lunch time, on every port from 1024 to 32767, and then gave up. The only positive outcome of this was that, during the attack, all of the other hackers were blocked by the limited remaining bandwidth.

It was obvious that something positive had to be done to stop this nonsense.

We decided to find an intrusion detection system which, everyone agreed, would solve our problem, and made a list of the functions we wanted it to perform.

First, it had to be content-based, so it could identify a hack attempt by the kind of thing the query was trying to do, which implied that such a system would need a certain amount of intelligence.

Second, having identified the hack, it would need to remember the IP address, drop the connection, and make sure that that IP address would never again be allowed to connect to our site.

Last, it would need to do all this in less than one second. The attacks that we faced were not directed from Mum and Dad's Wintel PC, but from high-end Unix servers in data centres. Having seen the speed at which our log monitor scrolled up the screen when we were under attack, we then examined the access log, and noticed that the average zombie hijacked server could shower us with hack vectors at a minimum rate of two or three a second and, sometimes, if they'd hacked a decent machine, up to ten a second.

Our goal was to stop it after the first vector.

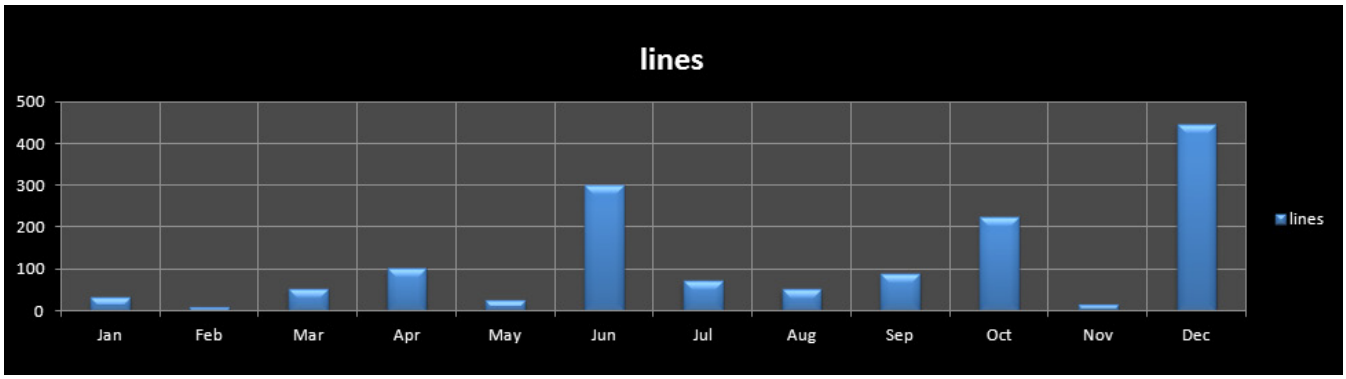


Figure 1. Before deployment of IDS

The search for the product

What we expected was, that we would make a quick list of suitable products, then spend a long period of decision-making, choosing from many suitable candidates. This turned out to be a huge disappointment.

We noticed from the first day, that the vast majority of intrusion detection systems were really no more than fancy java, shell and perl scripts, with a response time similar to that of a whale trying to turn itself around.

Disillusioned with the (not so) cheap end of the market, we decided that something used by banks had to be of the right quality, so we took a look at the professional, so-called ‘enterprise level’ products.

While researching this kind of product online, the whole thing got off to an unpromising start, when I read the comments of a security consultant to a bank, describing the product they used. During his speech, he declared proudly, that they would be aware of an intrusion within forty-eight hours of its happening.

Forty-eight hours? To us, forty-eight seconds would be too long, never mind forty-eight hours.

Predictably enough, the search of the high end of the market showed that shell scripts could be available at high prices, too.

Worse, most of this stuff only ran on Windows, and we’re a Sun Solaris shop. Who, in his right mind, would run a website on Windows?

During the demo of one of these products, the salesman explained that his system took its data via a network connection to the actual web server machine, and it had this absolutely mind-blowing graphical display of how your website was being hacked, minute by minute. This was impressive, and a lot easier than watching lines of text scrolling up the screen.

We asked how it worked, and were told that it counted the number of queries received in a given period and, if that exceeded a given value (which we could preset, of course) it flashed a lot of lights on the panel, and sounded an important alarm bell. Yes, but how did it differentiate between a legitimate connection, which just happened to be from a particularly fast machine, and a hack script? Well, it didn’t, but the final decision would be up to its operator. Did that mean that it didn’t automatically cut off the incoming connection? That’s correct. The system administrator would have to do that.

The salesman explained, rather frostily, that what we wanted was an intrusion protection system, not an intrusion detection system.

Since his product was totally unaware of the content of each query, we rejected it, and took a look at another, which claimed to be content-aware. This was more promising, since it was possible to pre-program the thing with a selection from a set of internally stored, popular hack strings, and have it do the usual light flashing and frantic beeping when it discovered something interesting.

Although it ran on Linux, and a source code licence was available (at an additional cost), so that we could recompile it to run on Solaris, it, too, relied on the system administrator to do something about the hacker. Furthermore, there was no provision for adding new hack strings to the list hard-coded inside it.

Further questioning revealed that the thing ran like a packet sniffer, and reassembled each packet’s payload to figure out the query string. This procedure resulted in many false positives, and false negatives, and made its response time less than breathtaking.

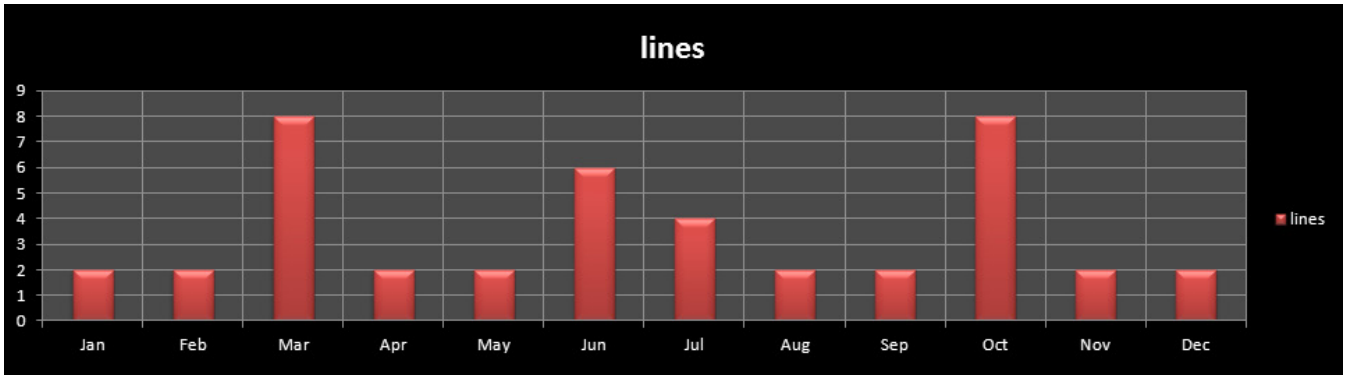


Figure 2. After deployment of IDS

The only product which, apparently, did what we wanted was a proxy. Filled with new enthusiasm, we took a cautious look at a few proxy offerings, only to be further disappointed.

Although a proxy really could do content-based filtering, accessing our web pages through it proved to be virtually impossible. Also, the degree of remote control available was strictly limited, to the point of being unusable.

So, there it was. The market was willing to sell us a few Linux offerings, a huge number of Windows ornaments, but nothing that would examine the content of what was trying to get into our website, and automatically drop the connection, if it saw something it didn't like.

The solution

During the time we were talking to the representatives of the various intrusion detection companies, I was thinking about the various issues which surrounded this problem.

Firstly, we absolutely needed to know the content of each query. However, there was no way that we would accomplish this reliably, by tracking text strings across several hundred packets, and then reassembling the original query. The packet stream contained too much information, some of which was irrelevant, and identifying the query with any degree of certainty was too difficult.

What we needed was a pre-assembled query, which was guaranteed to be a query.

I was sitting in front of the apache access log monitor, in the middle of a botnet attack, watching it scroll enthusiastically up the screen, when it occurred to me, that what happened at packet level was totally irrelevant. Bad things would only happen at the time when apache had the whole query in its buffer, and was about to act on it. Therefore, if we read the access log as it was being created, we would only be one line behind apache.

The hack queries themselves didn't bother us, since they attempted to exploit vulnerabilities in software we didn't use, so allowing one to slip through would be of no consequence.

So, the only criterion was to identify the first in a series of malicious queries from a given address, and do something before the hacker could send a second query.

Now the question arose, as to what constituted a malicious hack?

So, what is a hack vector?

We filtered our access log, and removed all queries which accessed our legitimate web pages and CGI executables. What remained, according to Sherlock Holmes, had to be the truth.

A lengthy and detailed examination of the logs showed a rich selection of attempted hacks.

One that was extremely prolific, was a GET followed by a series of '../../..' of varying lengths, terminating in some significant filename, like /etc/passwd. This would have to be the first on our list, since so many hackers, with no lnowledge of Unix, thought it had some chance of succeeding.

Next, we noticed blocks of up to a thousand hexadecimal characters, each preceded by a percent sign. Decoding these, revealed that they were either IP addresses, or filenames, which some incompetent hacker assumed would slip past the casual observer. This hack's secondary function was an attempted buffer overflow, caused by its sheer length. A definite second choice for blocking.

Almost identical in purpose, was a similar hack, but with the percent sign replaced with '\x'. However, the hexadecimal values weren't ASCII.
This was a puzzle, which took a lot of research, until I recognized one of the hexadecimal values as being the Intel processor opcode for 'CALL subroutine'. Hackers call these things shellcodes, and the

intent is to execute a buffer overflow, so they can place Intel machine code in the system's RAM, take over the CPU's program counter, index it to point to their own code, and execute anything they like on your machine. For any machine running on an Intel CPU, this would be the kiss of death.

With so many '\x' characters, this hack was easy to identify, so we added it to the list.

Then, there was the embedded question mark, usually followed by what looked like a script of some kind. and the embedded exclamation mark, usually in the middle of a lot of different hexadecimal stuff, which was obviously up to no good.

There were also the SQL injection hack attempts. I guess the most original, was one which attempted to overflow the CAPTCHA buffer (which we didn't use) with a script like this:

```
"captcha/img.php?code=1'%20AND%201=0%20UNION%20SELECT%20SUBSTRING(CONCAT(login,0x3a,passwo
rd),1,7)%20FROM%20User%20WHERE%20User_id="
```

We decided against wasting computing time on these, since our other criteria, such as the string '.php' and the percent signs, would easily identify it. Finally, there were quite a few hacks containing an embedded series of plus signs, usually accompanied by a string of hexadecimal, or plain text like 'Result:+no+post+sen ding+forms+are+found'. Just for the sake of complete coverage, we added a line of code to reject these.

Collating all of the information revealed something even more interesting. It became obvious that approximately ninety percent of all hack attempts of all kinds were aimed at dozens of different PHP files.

The attacks varied from simple GET queries and POST queries, to a pattern, where an initial query would attempt to GET a file like index.php (presumably, to establish its existence) and be followed by a second query, which would try to POST to the same file, and overwrite it with a back door. Then, a third query would try another GET.

In the light of these observations, we decided that another primary candidate for blocking would be any query containing the string '.php'.

Command and Control

What happens once the malware is installed on your computer?

Since Unix, unlike Windows, doesn't permit self-executing executables, the hacker needs to access his malware after it has been installed.

How is he going to do so? Any self-respective server will have all ports closed except port 80, and believe itself to be totally impregnable. Unfortunately, this is not the case, since it is through port 80 that the C&C will wake up and direct the malware.

Almost all security devices concentrate on monitoring and defending TCP traffic through port 80. The C&C, on the other hand, talks to the malware using the UDP protocol, also through port 80, and is invisible to apache, and to many security systems.

It's perfectly reasonable to block UDP traffic, with few resulting issues. However, just to complicate matters, there are other services, which run on UDP. DNS queries and replies, the Unix XDMCP login, and time server data are just a few examples. Any firewall rule which blocks UDP traffic, has to exclude these.

Dropping the connection

When we reached this point in the investigation, I could almost write the code for the content analyzer in my head, and it was beginning to look more and more possible that we could write our own intrusion detection system. Then, I thought about the tricky part: dropping the connection.

The first thing to come to mind was a utility called tcpkill, which will very nicely drop an established TCP connection. However, a moment's reflection showed that this would be inadequate. The average hack script re-sent the same line anything up to four hundred times and, if we invoked tcpkill every time, not only would the network traffic be no lighter, but the CPU would chase its tail trying to keep up with the repeated hack attempts as well, especially when handling an attack from a few dozen servers simultaneously.

The next thought was that we would use the firewall.

Since we expected that our IDS would be a stand-alone process, it would be necessary to use a firewall which was remotely programmable. Almost every supplier that we contacted claimed to have such a device, so things were looking very promising.

Unfortunately, firewalls are very security-conscious animals, and the only way to remotely program them, is to login to them first. The procedure for doing this was either through a gee-whiz graphical user interface, or via a telnet or SSH TCP connection. The GUI was obviously unacceptable, so we wrote piece of code which established a telnet connection to the firewall and sent it a new rule. Ten seconds later, it was back on line.

Most firewalls contain a minimal Linux computer, and every time a new rule is added, this computer is rebooted. Even though ten seconds is a very short time for a reboot, it was just too long for our purposes.

Apart from the huge delay to add another rule, during that ten seconds, the machine would be sitting there with open arms, welcoming all hackers to do their worst, since the firewall was resetting itself, and totally inoperative. Further, that ten seconds would allow several hundred new hack attempts to queue up for processing, resulting in a never-ending shuffle between our content analyzer and the firewall.

Firewalls were abandoned, and we turned our attention to the Unix operating system.

Solaris has am extremely powerful utility, called 'ipf', which is a version of the 'ipfilter' module, which dates back to SunOS 4.1.3, in the good old BSD days.

It has all of the facilities available in stand-alone firewalls, such as NAT, but the filtering is actually performed in the Unix kernel, making it extremely efficient. It gets its rule set from a file, which is a minor drawback, but I decided to try it, anyway.

I wrote another piece of code, which appended a new firewall rule to the file, then told ipf to re-read the file and restart. We ran a few tests, and found that the time delay was almost immeasurable.

This is actually not that surprising. Since the filtering is done in the kernel, there is no actual ipf process. When a user issues a command to re-read the configuration file, the kernel activates a 'read' system call, which is internal to itself, so there isn't even a separate process to re-spawn. The only delay, is the time taken to execute the disk I/O – which is always a high priority task, since the kernel knows it takes a long time.

We decided against including any facility to count the number of queries in a given time interval. If the purpose was to identify a DDOS, then the hardware firewall could adequately cope with it. Also, this would mean repeatedly stopping other processing for the duration of that time interval. This could add an order of magnitude to the response time.

The complete system

We now had all of the building blocks for a complete intrusion detection – or, more accurately, intrusion protection system.

On startup, the IDS would read the ipf configuration file, and store all of the rules in an array of data structures. This would put the IDS in sync with the firewall, which was necessary, so that we didn't try to add a rule for an IP address which was already being blocked.

Next, we called a function which opened the apache access log, and performed a seek to the last line in the file. Having done that, it entered an endless loop, and waited for another line to be added to the file.

The loop contained the code of the content analyser, and had no time delays or pauses built in, so it would execute as fast as the CPU could execute machine code. This is usually extremely bad practice, since it uses 100% of the CPU's processing power. In our case, it didn't matter, since our machine had 32 CPU's, and devoting one of them to the IDS was a good investment.

As soon as apache logged another query, the content analyzer would scan it to see if it contained any of the hack signatures which we had built into it. If a hack attempt was identified, a firewall rule would be automatically created, to make comparison with the stored firewall rules easier, then another function would be called, to see if that IP address was already being blocked.

If this turned out to be a new hack attempt, the ipf configuration file would be opened, the new rule appended, and ipf re-invoked so it could re-read the file. Having performed the most important operations, the IDS would then add the new rule to its internal store.

There is a great temptation, when designing a system like this, to use multi-threading, or parallel processing. Although this would have considerably speeded up part of the processing cycle, the dangers of collision, between threads or processes, in areas such as file reading or writing was too great. Semaphores and mutexes are traditionally used to obviate such problems but, in general, if you need to use a mutex, you're either doing it wrong, or you shouldn't be multi-threading.

Conclusion

After a short period of debugging, the IDS was commissioned, and we monitored its progress over the first week, or so.

The performance was even better than we expected, and there were no false negatives. Anything that was supposed to be stopped, was stopped dead, after apache received just one illegal query.

However, there were a number of false positives.

We examined the logs, and found that some query strings, especially those which were links from some online magazines, and some social media sites contained elements containing the string '.php'. This was enough to trip the content analyzer, and have the IP address blocked.

There were so few of these false positives, that we were willing to write this off as acceptable collateral damage, when compared with the enormous benefit of limiting each hacker to one hack attempt per lifetime. With the possible exception of LinkedIn, the social media sites were unlikely to bring us any significant business, but some of the online magazines were important. Accordingly, we added a few lines of code into the loop, which would cause it to ignore any positives containing the names of chosen sites.

So far, the IDS has been running continuously for over two years, with no modifications, apart from the periodic addition of new rules, as new hacks are discovered.

If this were a commercial product, we would probably have it read a configuration file on startup, instead of having the hacks and exceptions hard-coded. However, since it isn't, we don't mind recompiling it each time there's an update. It keeps it more secure.

About the Author

Design Simulation Systems Ltd
http://www.designsim.com.au
Consultant to Forticom Security
http://www.forticom.com.au

Bridging the gap between business & technology

Ranked among the largest minority-owned IT services firms in the U.S. having Global Delivery capabilities

Operations in the U.S., U.K., India, Singapore and Philippines with over 9000 professionals

Leading mid-tier IT vendor with end-to-end IT capabilities spanning Technology Consulting, Application Outsourcing and Infrastructure Services

Recognized on numerous occasions by GS 100, The International Association for Outsourcing Professionals, FinTech 100 & InformationWeek

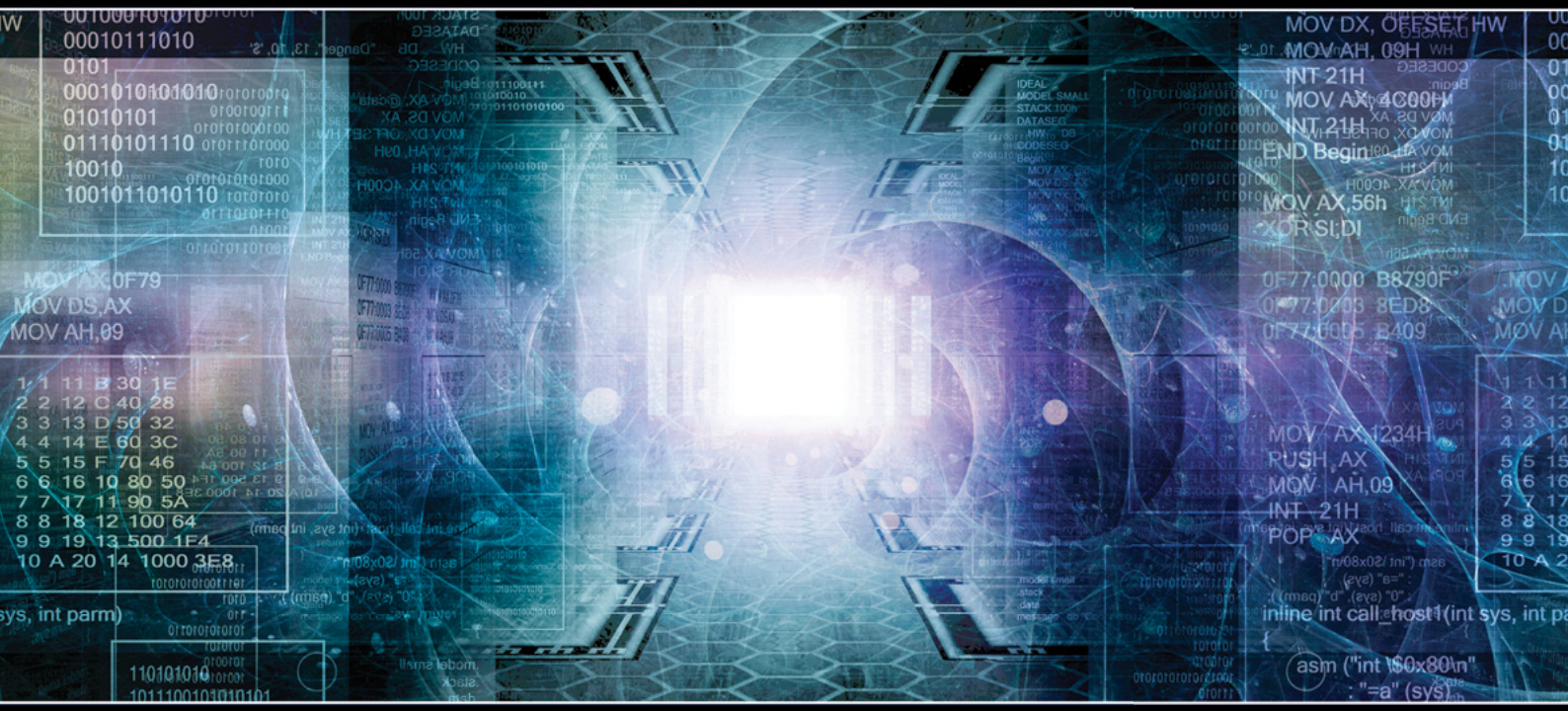
Collabera

www.collabera.com

ADVANCED TARGETED ATTACKS

HAVE PENETRATED **95%** OF ALL NETWORKS*.

THINK YOU'RE IN THE 5%?



You may think your existing security defenses prevent advanced targeted attacks from entering your network and stealing your data. They don't. Advanced attacks easily evade traditional and next generation firewalls, IPS, AV and gateways. Your best defense is **FireEye**. Trusted by the Fortune 500, and over 60 government agencies globally, FireEye is the leader in helping organizations combat advanced malware and targeted APT attacks.

Put a stop to advanced attacks with advanced security. Visit us today at **www.FireEye.com/StopAPTs** and let us help you close the hole in your network.

FireEye™

*Based on FireEye end-user data